Cloud Web Services Guide

Version DWS3
Sep 2022

docmosis

Cloud Web Services Guide

Table of Contents

1 Introduction

1.1 Overview

The Cloud Web Services Guide is intended for software application developers and integrators who need to produce formatted documents and reports from applications.

Docmosis Cloud Services provide an easy way to generate sophisticated and dynamic documents from virtually any application. The combination of web services and the Docmosis engine provides capability that can be integrated rapidly.

The Cloud Services are:

- *Template Driven* - you can change your templates any time with a word processor; upload and they will take effect immediately - wherever your application is running.
- *Accessible* - as long as you have internet connectivity you can render your documents using just about any development environment and delivered to multiple destinations. Processing can be focussed within geographical region to meet privacy and data-sovereignty obligations.
- *Secure* – all communications between Docmosis and your application are SSL encrypted and Docmosis doesn't hold your data or documents after processing.
- *Reliable* – multiple levels of redundancy have been built upon on the Amazon Web Services (AWS) platform, providing security and reliability. 24x7 internal and external monitoring integrated with Pingdom and Status.io allows customer visibility to Docmosis Cloud performance.
- *Flexible* - the Docmosis engine provides rich template capabilities and output formats.
- *Simple API* - calls to the service are made using HTTPS/SSL form posting. The *Render* service may be the only service that needs to be called. However, it is supported by other services, for example: to manage templates and images.

Templates can be uploaded and downloaded using either calls to the Cloud Service API or using the Cloud Console.

1.1.1 Terminology Used in this Document

Term	Definition	Term	Definition
Template	A normal Microsoft Word or LibreOffice document containing special Docmosis fields.	**Fields/ Placeholders**	Docmosis specific mark-up within the template that controls the insertion and removal of data and content.
Render	The process of merging data with a template to generate a document.	**Access Key**	A unique string of characters sent by the application calling the API to verify it has permission to use the service(s).

1.1.2 Related Reading

The *Cloud Template Guide* provides fundamental details on the creation of templates. Refer to this document to ensure the data you send to Docmosis matches the data required by the template.

1.2 Key Concepts

1.2.1 Processing Locations

There are multiple locations where the Docmosis Cloud services operate:

- United States
- Europe
- Australia

The services running at each of these locations are equivalent except that each location has it's own templates, images, storage and processing. Your account will have access to one or more of these locations, allowing you to:

1. Optimize performance by choosing a location which is closest to your applications. This will reduce the latency of communications.
2. Geo-bound your processing to a location as required.
3. Partition your application to provide different documents in different locations.

When calling the services from your code, or using the Docmosis Cloud Console, you will always be communicating with a specific Docmosis location.

1.2.2 Character Encoding

All data passed to the Cloud Services should be `UTF-8` encoded. This provides a great balance between flexibility and compatibility. If you pass data containing special characters, then you will need to ensure you are `UTF-8` encoding it, otherwise you'll get unexpected characters in your resulting documents.

1.2.3 Fonts in your Templates

The Cloud Service has most of the common and popular fonts.

Your templates should only use fonts that are available on the Cloud Service. If you use fonts which are not installed, you may see unexpected font substitutions in your PDF documents or inaccurate page references when using indexes or tables of content.

Contact Docmosis Support (support@docmosis.com) for a list of available fonts.

1.2.4 Dynamic and Stock Images

When Docmosis generates a document containing images, the images can be sourced in three different ways:

Sent with your data: To send images with your data, they should be Base64 encoded and included in your data like any other textual information.

See Including Base 64 Dynamic Image Data on page 20 for more information.

Sourced from files uploaded to the Cloud Service: “Stock” images are images which are uploaded to the Cloud Service and dynamically sourced and inserted during document generation. This is ideal for logos and signatures which change only occasionally or there is a set to select from.

Docmosis will retrieve the images when needed, so they don’t need to be streamed each time.

See Including "Stock" Image Data on page 20 for more information.

Sourced from a URL: Image data can also be dynamically sourced from URL references in your data. This means your data has a URL reference to an image and Docmosis fetches and inserts the image during document generation.

See Including Dynamic Image Data from URLs on page 20 for more information.

1.2.5 Document Storage

Docmosis has the ability to store your generated documents to Amazon Web Services S3. This is controlled using the `storeTo` parameter of the `render` service. By default, this option may be disabled, and needs to be enabled by our support team. Please contact support@docmosis.com for further details.

1.2.6 Template Merging

The render process is powerful enough to merge multiple templates into a single document. Templates may reference other templates dynamically (via data) or statically (in the template itself). This provides a mechanism for inserting common content across multiple templates.

See the *Cloud Template Guide* for information about how to reference one template from another.

1.2.7 Production vs Development Mode

Some services provide the option to operate in a forgiving manner (development mode) or in a very strict manner (production mode). The intention is that in development mode you are allowed to produce documents that contain errors, helping you to locate the error and make the necessary adjustments.

In production mode, no document with detected errors will be produced. Instead, the operation will fail with diagnostic information so you can be assured that documents will never be delivered that have fundamental errors in processing.

1.3 Troubleshooting

The FAQ section of the Docmosis Resources website (https://resources.docmosis.com) may help with troubleshooting problems when using the Docmosis Cloud Service API.

2 The Developer API

2.1 Fundamentals

Docmosis Cloud Services is a REST-based API. You can find more information about REST at: WikiPedia REST.

All calls to Docmosis are made using HTTPS POST requests. You can write code to call the API directly or use a third-party toolset like the Java Jersey Client (http://jersey.java.net) creating your own requests. There is example code in various languages available on the Docmosis web site.

Access and identification are provided by your API Key "`accessKey`". Your account may have multiple access keys and can be rotated and expired as required. An `accessKey` is required with each API call and can be a request parameter, or provided in the http header.

All API calls will use a location-based url. The available locations are:

Location	URL base
`United States`	https://us.dws3.docmosis.com/api/
`Europe`	https://eu.dws3.docmosis.com/api/
`Australia`	https://au.dws3.docmosis.com/api/

Each location is independent of the others and your account will have access to one or more location. Each location has its own templates and all processing is confined to that location to support data-sovereignty and privacy requirements as well as optimizing performance. Access keys are global and apply in all regions.

As an example, to upload a template to Docmosis in the US and then render it, you would use these two API urls:

https://**us**.dws3.docmosis.com/api/**uploadTemplate**

https://**us**.dws3.docmosis.com/api/**render**

Since each Docmosis location is independent of the others the templates must be in the same location as you are rendering.

2.2 Response Codes and Messages

For every call you make to the Cloud Service, you should first check the response code to determine whether the call succeeded or failed. Once you know whether the call succeeded or not, you can choose whether or not to check for further information in the response body.

The service returns status codes as follows:

Status Code	Definition
200	Successful operation
400	Your Docmosis request is not valid or indicates a not-ready status.
500	A server error has occurred
404	Invalid URL (not found)

Other 4** and 5** response codes may also occur. You should always confirm that you received a 200 response before assuming success.

The Cloud Service also returns information about the result in JSON or XML format as follows:

Field	Definition
Succeeded	"true" or "false".
shortMsg	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
longMsg	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

Each service may also return additional information in the response information, as indicated in the sections to follow.

In the case of some errors, the response body will NOT include the data structure containing “succeeded”, “shortMsg” etc. This is because some errors indicate the call failed to even reach the Cloud Service. For example, a status code 404 indicates a bad URL in which case the status code alone is enough to indicate the nature of the problem.

2.3 A Quick Tour of the API

The API is divided into 4 areas:

1. Rendering – the most important service.

2. Template Management – uploading, downloading, listing templates.
3. Stock Image Management – uploading, downloading, listing images.
4. Document / File Storage Management – up/downloading, listing files.

Each of these areas is detailed in the following sections.

2.4 The Render Service

The `render` service is the document production 'work-horse', and it is typically the main service you need to call from your application, since template operations may also be carried out via the Cloud Console. You can call the Render service with data and instructions indicating: which template to use, the formats you require, where to send the result, and more.

Render works in production mode by default, meaning that any errors in the template or data supply are considered fatal and the render call will fail. You may override this with the `devMode` flag.

2.4.1 Service URL

`/render`

`/renderForm` (use this when data items are posted as key/value pairs in the API call, rather than in a "data" element).

2.4.2 Content-Type

There are different ways to call the render service based on `content-type`. Set the `content-type` in your request as follows:

Content Type	Description
`Multipart/form-data`	Parameters are passed as separate form parameters. The data parameter may be either XML or JSON. The "data" parameter may be omitted if using the `renderForm` end point.
`application/x-www-form-urlencoded`	Parameters are passed as separate form parameters url-encoded. The data parameter may be either XML or JSON. The "data" parameter may be omitted if using the `renderForm` end point.
`application/xml`	A single XML document string provides instructions

Content Type	Description
	and data (see examples below).
`application/json`	A single JSON document string provides instructions and data (see examples below).

Choose the `Content-type` that is easiest for you to work with.

2.4.3 Request Parameters

There are many parameters to control the render method, but most are optional. Please see the details in the table below for each parameter.

As an example, using the `application/json` content type a simple JSON format request could look like this:

```
{"accessKey":"XXXXX",
 "templateName":"template1.doc",
 "outputName":"result.pdf",
 "data":{"title":"Company Profile",
         "scope":"Initial Report"}}
```

You can see the data and instructions are combined into a single JSON structure.

The same request in XML format would look like:

```
<?xml version="1.0" encoding="utf-8"?>
<render accessKey="XXXXX" templateName="template1.doc"
outputName="result.pdf">
  <data>
    <report title="Company Profile"
            scope="Initial Report"/>
  </data>
</render>
```

The table below details the settings and options for the render request.

Parameter (bold=mandatory)	Description	Default
`accessKey`	Your unique access key that you can find on the Account page in the Cloud Console. Note the accessKey can be alternatively provided using a http header.	
`templateName`	The name of the template to use. The template must have been uploaded previously with the template upload request or via the Cloud Console.	

Parameter (bold=mandatory)	Description	Default
	Multiple templates can be specified using the ; character to separate the templates. With multiple template names, a single `outputName` for PDF output results in a single combined PDF. In all other cases, a ZIP file will be returned containing each rendered document.	
`outputName`	The name to give the rendered document. If no format is specified (see `outputFormat`), the format of the resulting document is derived from the extension of this name. For example, "resume1.pdf" implies a PDF format document. The name may be supplied without an extension (eg "resume1") and the `outputFormat` parameter will specify the format(s) to return. If multiple template names are specified, multiple (matching) output names can be specified to create a zip of named outputs.	
`outputFormat`	The format(s) of the rendered document. This can be a single format, or a ';' semi-colon delimited list. Multiple formats imply a zip file and `outputName` will have .zip appended as required. Files inside zip will be named using `outputName` and will have the format-specific extension appended as required. Valid options are pdf, doc, odt, rtf, html, txt.	
`storeTo`	Specify where to send the resulting document. If no specification is given, "stream" is assumed and the result will be streamed back to the requester, otherwise the ';' semi-colon delimited list of destinations will receive the result. Valid options are `stream, mailto, s3`. See section 2.4.4 StoreTo Options for more details.	`stream`
`compressSingleFormat`	Optionally choose to zip the result when a single output document is produced. The zip archive will contain a document in the specified format with a name based on `outputName + outputFormat`. The resulting zip file name will be the `outputName` with the .zip extension appended as required. This option is ignored if more than one `outputFormat` is specified. Positive values are "y", "yes" and "true" (case-insensitive).	`false`
`devMode`	The render service can be used in development or production modes respectively. If set to "y", "yes" or "true" this operation will work in "dev" mode, meaning that if something is incorrect in the template, data or instructions, Docmosis will do its best to produce a	`false`

Parameter (bold=mandatory)	Description	Default
	document. Such a document may contain errors such as missing images and data, and wherever possible, Docmosis will highlight problems to indicate the failure. In production mode, errors in document rendering will result in a failure result only, and no document will be produced. The production mode is to ensure that a bad document is never produced/delivered to a recipient. The default mode is production (that is, `devMode` is off).	
`data`	The Data that will populate the document. This may be either XML or JSON format. The type of data given determines the format of the response. If you call the "`/renderForm`" variant of the render end-point (instead of "`/render`") the data field is not required. Instead data will be extracted from the submitted form parameters using the parameter name as the key.	
`mailSubject`	If sending email, this will be used as the subject line of the email.	
`mailBodyHtml`	If sending email, this will be used as the body of the email and will be sent as html format.	
`mailBodyText`	If sending email, this will be used as the body of the email and will be sent as text.	
`mailNoZipAttachments`	If this is set to true, any email attachments will be attached as individual files rather than as a single zip (when multiple formats are being used).	`false`
`mailGatewayName`	The custom mail gateway name to use. If using a "mailTo" `storeTo` directive then emails will be sent via this gateway. Note that the gateway must be preconfigured in your account for your chosen Environment.	
`requestId`	Any string you would like to use to identify this job. This string will be returned in responses.	
`sourceId`	Any ID you would like to associate with this render. This could be a device id, a project code or any meaningful piece of data you associate with this render. This value can be reported later with associated monthly counts. Limited to 150 characters.	
`stylesInText`	If set to "y","yes" or "true", your data will be parsed	`false`

Parameter (bold=mandatory)	Description	Default
	looking for html-like mark-up. The following mark-up is supported: - Bold e.g. `"this is <b>bold</b>"` - Italics e.g. `"this is <i>italics</i>"` - Underline e.g. `"this is <u>underline</u>"` - Cell Colouring e.g. `"<bgcolor="#ff0000"/>"` The `bgcolor` mark-up must be at the beginning of your data and the template field must be inside a table-cell to take effect.	
`passwordProtect`	If specified, this parameter will set the password for PDF and DOC files created by the render. The password is used when opening the document. Use with care as setting the password means the recipient must know the password to read the document. Note: `pdfArchiveMode` will disable any password setting for PDF documents.	
`pdfArchiveMode`	Create pdf documents in PDF-A mode for long term storage. Note this setting disables certain PDF features such as password protection and external hyperlinks.	`false`
`pdfWatermark`	If specified, PDF documents will have the specified text added as a watermark across the document.	
`pdfTagged`	If specified, the PDF documents will have extra information inserted to assist with low-vision readability tools. The alt-text for images in particular becomes "readable".	`false`
`pdfRestrictPassword`	Set the password for further security restrictions. Setting this also enables the further restrictions.	
`pdfRestrictPrinting`	Restrict printing of the created PDF: 0 = cannot be printed 1 = print only low resolution 2 = print in full quality Requires that pdfRestrictPassword has been set.	`2`

Parameter (bold=mandatory)	Description	Default
pdfRestrictEditing	Restrict editing of the created PDF: 0 = no edits allowed 1 = pages can be inserted and rotated 2 = form fields can be filled in 3 = form fields can be filled and comments can be added 4 = above 1-3 enabled, but page extraction disabled Requires that pdfRestrictPassword has been set.	4
pdfRestrictCopy	Set whether copy of PDF content is disabled. "true" or "y" disables the copy. Requires that pdfRestrictPassword has been set.	
pdfRestrictAllowAccessibilty	Set whether content can be extracted by accessibility applications. "true" or "y" disables the accessibility access. Requires that pdfRestrictPassword has been set and also pdfRestrictCopy must be set.	
ignoreUnknownParams	If true, unknown parameters in the request are allowed and ignored. By default, the render service will return an error if a parameter is specified that is not expected (defined by this table).	false
tags	A semi-colon delimited list of tags to record against this render. The tags can be later queried (using the getRenderTags end point) to retrieve stats such as page-counts and document-counts related to the tags.	
streamResultInResponse	If set to "y", "yes" or "true", the streamed result will be base64 encoded and included in the JSON or XML response under the key "resultFile". Note this only applies if the request includes (or implies) a "stream" result (see the storeTo parameter above).	false

2.4.4 StoreTo Options

Docmosis can render to several destinations at once, and optionally send different formats for delivery to each destination. As a simple example:

```
stream:pdf;mailto:doc
```

This indicates a PDF document should be streamed back to the caller, and a DOC document should be emailed.

By default, all destinations will receive all formats specified by `outputFormat` (or implied by the `outputName` if `outputFormat` not specified). Each destination may override the defaults settings and specify what to receive using this style "`stream:<format>`" e.g. "`stream:pdf`". If you wish to specify multiple email addresses, use multiple `mailto:` directives. Note that email behaviour is also determined by other parameters in the render call such as subject and body message.

The following table describes the available storage options.

Destination	Examples
`stream`	Stream the document back to the caller. By default this will be a binary stream direct response. If "`streamResultInResponse`" is specified in the request, the document will be base64 encoded and included in the JSON or XML response instead under the key "`resultFile`".
`mailto`*	Email* the document to the specfied address.
`S3`	Store the document in an external Amazon Web Services S3 location. This option is disabled by default. Please contact our support team to enable and configure this option for you.

The following table provides some examples.

Destination	Examples

`stream`	`stream`
	`stream:pdf`
	`stream:pdf,doc`

`mailto`*	`mailto:support@docmosis.com` `mailto:support@docmosis.com:pdf`
`S3`	`s3:my.amazon.bucket,mydocument1.pdf` `s3:my.amazon.bucket,mydocument1.zip:pdf,odt`

* Note: email delivery is not guaranteed. It is generally less secure, less reliable and slower than other delivery mechanisms since much of the delivery process is outside of the control of Docmosis.

The storage destinations may be repeated as required. For example, multiple emails can be sent by specifying `mailto:address1@my.com;mailto:address2@my.com`.

2.4.5 Including Base 64 Dynamic Image Data

Image data can be included in the data stream. This is achieved by Base64 encoding the image data and assigning the value to the key which your template image is using. The image data (i.e. its value) must be prefixed by "`image:base64:`" so that Docmosis can identify and decode it as required.

As an example, an image in a template marked with "img_pic1" expects to find an image called pic1 specified in the data. In JSON format it might look like:

```
"data":{"pic1":"image:base64:mawv0dga423g0345....." …
```

Base64 encoding is outside the scope of this guide, but it is easy to find libraries and reference material to help you create it.

Image data is typically large compared with textual information. Keep in mind the impact on your bandwidth and document size when using image data. If there are only a few options for an image, consider using different templates, sub-templates or separately uploading "stock" images.

2.4.6 Including Dynamic Image Data from URLs

Image data can also be dynamically sourced from URL references in your data.

As normal, your template would have marked up the image with a name that ties to your data, for example “pic1”. To dynamically replace the image “pic1” with an image from a URL, the data would look something like:

```
"pic1":"[imageUrl:http://image.site/image/Image103.png]"
```

The above data would prompt Docmosis to fetch the image from:

```
http://image.site/image/Image103.png
```

and put it into the document dynamically.

Due to the potential impact on the Cloud platform, the use of URLs requires Docmosis support staff to white-list the URLs on your account.

2.4.7 Including "Stock" Image Data

Where the same image is repeated in document production, such as logos or signatures, you have options about how to obtain the image:

1. stream the image every time you render – this is wasteful of processing and bandwidth if the image is repeated.

2. put all the options for the image into the template then have Docmosis dynamically strip out the undesired image(s) during the document render. This can be done using conditional sections (See the *Cloud Template Guide* for more information).
3. upload the images in advance to your Docmosis account - these are called "stock" images. You can reference these images in your data, providing an efficient way to insert images into documents.

To use a stock image, you will need to upload it first. This can be done by logging into the Cloud Console and selecting Upload Image on the Images page. You can also use the API to upload images programmatically – see The Image Upload Service in section 2.14 on page 39.

Once your image has been uploaded to the cloud, you can reference it in your data using a key that matches your template image, and a specially formatted value. For example, if your template has the image named `img_pic1` and you've uploaded `face1.jpg`, your key is `pic1` and your value is "`[userImage:face1.jpg]`". In JSON format, your data would look something like this:

```
"pic1":"[userImage:face1.jpg]"
```

When you upload an image, you may also use a path-like structure for organizing your images. For example, you may have uploaded the image with the name:

```
projectA/first/face1.jpg
```

in which case, the request above would look like this:

```
"pic1":"[userImage:projectA/first/face1.jpg]"
```

2.4.8 Response Messages

The response from the render method varies depending on:

1. whether it succeeds or fails
2. whether your destinations include streaming back in your request.

Remember, you should always check the status code first to determine what to do next. Any status other than **200** means the render failed, and error information will be available in the response body.

The following cases show the types of check you should perform to extract the response information:

1. **On Success (status code = 200) and `storeTo` includes "stream":**

the body of the response is the binary document stream.

2. **On Success (status code = 200) and `storeTo` excludes "stream":**

the body of the response is a JSON object containing:

Field	Definition
`succeeded`	"true".
`requestId`	The `requestId` given in the render request (if any).

3. **On failure (status code <> 200):**

the body of the response is a JSON object containing:

Field	Definition
`succeeded`	"false"
`shortMsg`	A short message about the cause of the failure.
`longMsg`	A more descriptive message about the failure if applicable. It may be blank.
`requestId`	The requestId given in the render request (if any).

2.4.9 Response Headers

For the render service, the following headers may be returned to assist response processing.

Header	Notes
`X-Docmosis-Server`	An identifier for this service “dws3”.
`X-Docmosis-RequestId`	Returned if the original request provided a requestId.
`X-Docmosis-PagesRendered`	The number of pages rendered
`X-Docmosis-Document-Errors-Detected`	Whether or not errors were detected in the document rendering. This is useful in “dev mode” to be able to quickly determine whether the document created is known to have errors.
`X-Docmosis-Zip-Created`	Set to true if the result being returned is a zip file.

2.5 The Upload Template Service

This service allows you to upload a template to be (later) rendered into documents. When you upload a template, Docmosis analyses it and will report any errors in the template at this time.

Docmosis has a special capability to report errors in the template within rendered documents. This means that when you render a document you can see the error AND its location in the template. By default this capability is enabled (`devMode=true`) to assist with development. Any template uploaded in development mode with errors will not render unless the render also uses development mode.

> **Uploading of templates can also be done via the Cloud Console.**

Templates can also be uploaded in bulk using The Upload Template Batch Service detailed in section 2.11.

2.5.1 Service URL

```
/uploadTemplate
```

2.5.2 Content-Type

The content-type for the upload is "`multipart/form-data`".

2.5.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateFile`	The template file you are uploading.
`templateName`	An optional, overriding name, which may include a path.
`templateDescription`	A description for the template.
`devMode`	If set to "y", "yes" or "true" the upload is run in developer mode, meaning that if errors are detected in the template the upload still succeeds. This allows later render requests to optionally display the errors in the template in a rendered document (using the render service's own devMode flag). In a production setting, a devMode of true for upload is perfectly valid since the devMode of the render service has the final say as to whether a document is produced.

Field	Definition
	Defaults to "true".
`keepPrevOnFail`	If set to "y", "yes" or "true" the previous template of the same name will be left in place if the uploaded template has errors. If not specified (or "n", "no" or "false") the original template is always removed, even if this uploaded template has errors. This only has effect when `devMode` is disabled (since `devMode` is intended to allow templates with errors to be displayed by Docmosis). This parameter means that in production mode (non-developer mode) template uploads will not replace a working template with a bad template. Defaults to "false".
`fieldDelimPrefix`	If using plain text mark-up in your templates this specifies the prefix delimiter identifying a field. The default is '<<'.
`fieldDelimSuffix`	This is the closing delimiter for plain text fields. The default is '>>'.
`normalizeTemplateName`	If set to "y", "yes" or "true" the template name given will be NFC normalized (Unicode NFC normalization). The default is false.

2.5.4 Response Messages

The response is an indicator of success or failure plus any further helpful information. When the upload is successful, details are returned under the `templateDetails` key.

Field	Definition
`succeeded`	"true" or "false"
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
`templateDetails`	The details of the template in JSON: (when succeeded = true) - `name` - the template file name - `lastModifiedMillisSinceEpoch` - last modified in

Field	Definition
	milliseconds - `lastModifiedISO8601` - last modified yyyy-MMdd'T'HH:mm:ssZ - `sizeBytes` - the size in bytes - `templatePlainTextFieldPrefix` - the prefix used when it was uploaded - `templatePlainTextFieldSuffix` - the suffix used when it was uploaded - `templateDevMode` - the dev mode setting used when it was uploaded - `templateHasErrors` - true if the uploaded template has errors - `templateDescription`- the description uploaded with the template if any - `md5` - the md5 hash code for the template

2.6 The Get Template Service

Get Template retrieves the template that was originally uploaded.

Multiple `templateName` parameters can be specified to download multiple templates in one zip response (up to 100 in one request).

2.6.1 Service URL

```
/getTemplate
```

2.6.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.6.3 Request Parameters

Field	Definition
`accessKey`	Your access key.

Field	Definition
templateName	The name of the template. This parameter can be specified multiple times to download multiple templates (up to 100) in a zip file.

2.6.4 Response Messages

On success (status=200), the body of the response will contain the binary stream for the template.

On failure, the response provides the following information:

Field	Definition
succeeded	"true" or "false"
shortMsg	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
longMsg	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.7 The Get Template Details Service

Get Template Details retrieves the details of a template that was originally uploaded, but not the template itself.

2.7.1 Service URL

```
/getTemplateDetails
```

2.7.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.7.3 Request Parameters

Field	Definition
accessKey	Your access key.

Field	Definition
templateName	The name of the template.

2.7.4 Response Messages

On success (status=200), the body of the response will contain the data structure below.

On failure, the response will contain at least the succeeded and shortMsg fields.

Field	Definition
succeeded	"true" or "false"
shortMsg	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
longMsg	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
templateDetails	The details of the template in JSON: name - the template file name lastModifiedMillisSinceEpoch - last modified in milliseconds lastModifiedISO8601 - last modified yyyy-MM-dd'T'HH:mm:ssZ sizeBytes - the size in bytes templatePlainTextFieldPrefix - the prefix used when it was uploaded templatePlainTextFieldSuffix - the suffix used when it was uploaded templateDevMode - the dev mode setting used when it was uploaded templateHasErrors - true if the uploaded template has errors templateDescription- the description uploaded with the template if any md5 - the md5 hash code for the template

2.8 The Get Template Structure Service

Get Template Structure retrieves the structure of a template that has been uploaded. The structure returned describes fields, repeating and conditional sections etc. The primary purpose of this method is to allow automated processing based on what is currently in a template (such as creating dynamic data forms etc).

2.8.1 Service URL

```
/getTemplateStructure
```

2.8.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.8.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateName`	The name of the template.

2.8.4 Response Messages

On success (status=200), the body of the response will contain the data structure below.

On failure, the response will contain at least the succeeded and shortMsg fields.

Field	Definition
`succeeded`	"true" or "false"
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
`templateStructure`	A JSON format description of the template structure. Template elements are returned in an array, for example: “templateStructure”:[

Field	Definition

	{"type":"field", "fieldIdx":0, "text":"name", "dataRefs":["name"]},
	{"type":"field", "fieldIdx":1, "text":"address", "dataRefs":["address"]}
	]
	The above example show two fields were found and each correlates with a lookup on a single data item (“dataRefs”). A template element such as an expression-field may correlate with multiple data references. For example, the template expression:
	<<{firstName + ‘ ‘ + lastName}>>
	is reported as a field with two dataRefs:
	{"type":"field", "fieldIdx":2, "text":"{firstName + ' ' + lastName}", "dataRefs":["firstName","lastName"]}
	
	The types reported are: “field”, “repeat”, “condition”, “image”, “templateRef” corresponding to matching structures in the template. Each type has it’s own index which is global to the document. “text” shows the string as presented in the template and dataRefs reports the identified data elements correlated to the template element.
	The result also indicates the nesting of elements. The template content:
	<<cs_hasPeople>>
	<<rs_people>>
	<<name>>
	<<es_>>
	<<es_>>
	Would be expressed as:
	[
	{
	"type": "condition", "conditionIdx": 0, "text": "cs_hasPeople",
	"dataRefs": ["hasPeople"],
	"contains": [
	{
	"type": "repeat", "repeatIdx": 0, "text": "rs_people",
	"dataRefs": ["people"],
	"contains": [
	{
	"type": "field", "fieldIdx": 0, "text": "name",
	"dataRefs": ["name"]
	}
	]
	}
	]
	}
	]

2.9 The List Templates Service

List templates lists the templates available to you.

2.9.1 Service URL

```
/listTemplates
```

2.9.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.9.3 Request Parameters

To list templates, you need only supply your access key.

Field	Definition
`accessKey`	Your access key.
`includeDetail`	Include extra detail about parameters. Default=true.
`folder`	Limit processing to the given folder. Optional. The returned names are always relative to the folder.
`includeSubFolders`	Include the contents of sub folders (ie work recursively). (folder names are always included at the current level being listed regardless of this setting). Default=true
`paging`	Whether or not to return results in pages. Default=false. If true, pages of 1000 records are returned.
`pageToken`	When paging is true, this token identifies the next page to retrieve. The page token is null for the first page. When the first page response returns, it contains the token required to request the next page.
`pageSize`	Specify the size of pages. Defaults to 1000 but can be reduced.

2.9.4 Response Messages

The response includes the normal success indicator and messages as well as a JSON object containing the list of templates.

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
`templateListStale`	"true" or "false". If Docmosis detects changes to the template list are in progress (such as updates or deletions) this flag will be set to "true" to indicate the list is not necessarily up to date. This is only ever expected to be "true" for a short period after deletes or updates.
`templateList`	The list of templates in JSON format having attributes for each template: name - the template file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `templatePlainTextFieldPrefix` - the prefix used when it was uploaded* `templatePlainTextFieldSuffix` - the suffix used when it was uploaded* `templateDevMode` - the dev mode setting used when it was uploaded* `templateHasErrors` - true if the uploaded template has errors* `templateDescription` - the description uploaded with the template * `md5` - the md5 hash code for the template. * *These items are only returned when the includeDetail parameter was specified as true.
`nextPageToken`	If returned, this specifies the token to use in the next call to

Field	Definition
	listTemplates (parameter pageToken) to get the next page of results. This element is not present if the request was not-paged or there are no further pages.
`pageSize`	The size of pages when paging is active. Default is 1000 (which is also the maximum supported). Since the list returns folder names as items, the folders themselves count in the page size.

The `templateList` is an array of objects giving details for each template in the list.

2.10 The Delete Template Service

The delete template service deletes the specified template. Multiple `templateName` parameters can be specified to delete multiple templates.

2.10.1 Service URL

```
/deleteTemplate
```

2.10.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.10.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateName`	The name of the template. This parameter can be specified multiple times to delete multiple files.

2.10.4 Response Messages

The service responds with a simple indication of success or failure using the standard structure:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this

Field	Definition
	will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.11 The Upload Template Batch Service

This service allows you to upload multiple templates contained in a Zip file.

The zip file may be a simple list of templates or may have templates organized into folders. The folder structure of the zip file will be replicated in your cloud account. The upload is additive meaning the templates will be added to those already in your account and will overwrite any existing template in the same location with the same name.

This service, unlike most other services, runs asynchronously. A "job" is launched to perform the processing which could potentially take significant time. The caller can then check on the status of the job or cancel it using the appropriate end points described below.

Jobs are assigned a `userJobId` either by the caller or automatically. This id can be used to query the status of the job or to cancel a running job.

Uploading of templates can also be done via the Cloud Console.

In terms of concurrent batch uploads, the following rules apply:

- Uploads to different Docmosis Regions may occur concurrently (since the templates are independent) as long as you don't have another job running with the same userJobId (if specified).
- Uploads to the same region may occur concurrently provided:
    - The userJobId (if provided) is unique against any other currently running job (that you are running in any region)
    - There is no overlap to where the templates are being stored (ie the explicit or implied intoFolder).

Limitations to the size of the upload and the number of templates apply. Contact Docmosis support if you expect to upload zip files larger that 50Mb or with more than 1000 templates.

The zip is assessed in step one and the process will be terminated before any templates are affected if a problem with the zip is detected. The following tests are included: zip too large or with too many entries, any entries with invalid paths (references to drives or parent folders), any entries that look like scripts, executables, archives, etc.

2.11.1 Service URL

```
/uploadTemplateBatch
```

2.11.2 Content-Type

The content-type for the upload is "`multipart/form-data`".

2.11.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateZip`	The template files you are uploading. This can include folders which will be created in your cloud account. The zip is discarded at the end of the process, only the uploaded templates persist.
`userJobId`	An identifier for the batch upload job which can be used to check the status of the job or to cancel the job. If no id is provided a unique one will be generated and returned in the response message. Maximum of 40 characters. The jobId can be reused. When checking on the status of a job, only the latest version of the job with the id is considered.
`intoFolder`	The path to the folder to upload the templates to. If no path is provided the templates will be uploaded to the 'root' directory of your cloud account.
`devMode`	If set to "y", "yes" or "true" the upload is run in developer mode, meaning each template will be uploaded in Developer mode (errors allowed) and the job will complete if possible. Defaults to "true".
`keepPrevOnFail`	If set to "y", "yes" or "true" the previous template of the same name will be left in place if the uploaded template has errors. If not specified (or "n", "no" or "false") the original template is

Field	Definition
	always removed, even if this uploaded template has errors. This only has effect when `devMode` is disabled (since `devMode` is intended to allow templates with errors to be displayed by Docmosis). This parameter means that in production mode (non-developer mode) template uploads will not replace a working template with a bad template. In a batch of templates, this setting will only affect the last template processed because a failure processing the template will abort the batch job. Defaults to "false".
`fieldDelimPrefix`	If using plain text mark-up in your templates this specifies the prefix delimiter identifying a field. The default is '<<'.
`fieldDelimSuffix`	This is the closing delimiter for plain text fields. The default is '>>'.
`normalizeTemplateName`	If set to "y", "yes" or "true" the template names will be NFC normalized (Unicode NFC normalization). The default is false.

2.11.4 Response Messages

The response is an indicator of success or failure plus any further helpful information. When the upload is successful, job details are returned under the `jobStatus` key.

Field	Definition
`succeeded`	"true" or "false"
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
`jobStatus`	The status of the job in JSON: (when succeeded = true) - `userJobId` – An identifier for the batch upload job which can be used to check the status of the job or to cancel

Field	Definition
	the job. If no id was provided in the request a generated one is returned. - `isEnded` – "true" or "false". - `status` – The current status of the job. - `type` – The type of job being run. - `processingMsg` – A message about the template processing. - `startedTime` – Started time in milliseconds (Epoch time). - `finishedTime` – Finished time in milliseconds (Epoch time). 0 if job is ongoing. - `duration` – Duration of job in milliseconds - `pctComplete` – Percentage complete, 0 – 100.

2.12 The Upload Template Batch Status Service

This service reports on a currently or previously running template batch upload job. In the response, the "jobStatus" payload contains multiple items to establish the status of the job. Perhaps the most important data items for managing the job are "isEnded" and the "errorsDetected" field of the "jobResult" field.

When the job is completed, successfully or not, the jobStatus will contain the jobResult which has the details of the level of success of the entire batch.

2.12.1 Service URLs

```
/uploadTemplateBatchStatus
```

2.12.2 Content-Type

The content-type for the upload is "`multipart/form-data`".

2.12.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`userJobId`	The identifier for the batch upload job.

Field	Definition

2.12.4 Response Messages

Field	Definition
`succeeded`	"true" or "false"
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

`jobStatus`	The status of the job in JSON: (when succeeded = true)
	- `userJobId` – An identifier for the batch upload job which can be used to check the status of the job or to cancel the job. If no id was provided in the request a generated one is returned.
	- `isEnded` – "true" or "false".
	- `status` – The current status of the job.
	- `type` – The type of job being run.
	- `processingMsg` – A message about the template processing.
	- `startedTime` – Started time in milliseconds (Epoch time).
	- `finishedTime` – Finished time in milliseconds (Epoch time). 0 if job is ongoing.
	- `duration` – Duration of job in milliseconds (Epoch time).
	- `pctComplete` – Percentage complete, 0 – 100.
	- `jobResult` – The result of the job in JSON (when isEnded = true)
	  - `errorsDetected` – true if any of the uploaded templates have errors
	  - `uploadedIntoFolder` – the provided folder path to upload the templates to.
	  - `devMode` – the dev mode setting used when it was uploaded.
	  - `uploadedTotalCount` – total number of templates

Field	Definition
	uploaded. - `processedWithErrorsCount` – number of uploaded templates processed with errors. - `processedWithoutErrorsCount` – number of uploaded templates processed without errors.

2.13 The Upload Template Batch Cancel Service

This service allows a currently running template upload batch job to be cancelled. This service will return failure only if there was an error cancelling the job. If the job is already completed (so the cancel has no effect), this will return success and the message will indicate that it was already finished.

2.13.1 Service URLs

```
/uploadTemplateBatchCancel
```

2.13.2 Content-Type

The content-type for the upload is "`multipart/form-data`".

2.13.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`userJobId`	The identifier for the batch upload job.

2.13.4 Response Messages

Field	Definition
`succeeded`	"true" or "false" If the job has already completed, "true" will be returned. "false" is only returned if there was an error attempting to cancel a running job or the userJobId is invalid.
`shortMsg`	A short message about the result. In the case of an error this

Field	Definition
	will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.14 The Image Upload Service

The Image Upload service works the same as the template upload service, but is significantly simpler since there are few options to set.

Image uploading is used to create "stock" images that your data can reference when generating documents and Docmosis will take care of placing them into your document. This saves bandwidth and time when rendering the same images frequently. An alternative to uploading "stock" images is to place the images into your template and selectively filter them out using conditional sections.

2.14.1 Service URL

```
/uploadImage
```

2.14.2 Content-Type

The content-type for the call may be "`multipart/form-data`".

2.14.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`imageFile`	The file stream of the image.
`imageName`	An overriding name for the image which may include a path (e.g. `project1/stock/logo1.jpg`).
`imageDescription`	A short description for the image.
`normalizeImageName`	If set to "y", "yes" or "true" the image name given with be NFC normalized (Unicode NFC normalization). The default is false.

2.14.4 Response Messages

The response is an indicator of success or failure plus any further helpful information. When the upload is successful, details are returned under the `imageDetails` key.

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
`imageDetails` (when succeeded = true)	The details of the image in JSON: `name` - the image file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `md5` - the md5 hash code for the image

2.15 The List Images Service

List images lists the images available to you.

2.15.1 Service URL

```
/listImages
```

2.15.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.15.3 Request Parameters

To list images, you need only supply your access key.

Field	Definition
accessKey	Your access key.
folder	Limit processing to the given folder. Optional. The returned names are always relative to the folder.
includeSubFolders	Include the contents of sub folders (ie work recursively). (folder names are always included at the current level being listed regardless of this setting). Default=true

2.15.4 Response Messages

The response includes the normal success indicator and messages as well as a JSON object containing the list of images.

Field	Definition
succeeded	"true" or "false".
shortMsg	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
longMsg	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
imageListStale	"true" or "false". If Docmosis detects changes to the image list are in progress (such as updates or deletions) this flag will be set to "true" to indicate the list is not necessarily up to date. This is only ever expected to be "true" for a short period after deletes or updates.
imageList	The list of images in JSON format having attributes for each image: name - the image file name lastModifiedMillisSinceEpoch - last modified in milliseconds lastModifiedISO8601 - last modified yyyy-MM-dd'T'HH:mm:ssZ sizeBytes - the size in bytes md5 - the md5 hash code for the image

The `imageList` is an array of objects giving details for each template in the list.

2.16 The Delete Image Service

The delete image service deletes the specified image. Multiple `imageName` parameters can be specified to delete multiple images.

2.16.1 Service URL

```
/deleteImage
```

2.16.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.16.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`imageName`	The name of the image. This parameter can be specified multiple times to delete multiple files.

2.16.4 Response Messages

The service responds with a simple indication of success or failure using the standard structure:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.17 The Get Image Service

Get image retrieves the image that was originally uploaded. Multiple imageName parameters can be specified to download in a zip file (up to 100).

2.17.1 Service URL

```
/getImage
```

2.17.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.17.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`imageName`	The name of the image. This parameter can be specified multiple times to download multiple images (up to 100) in a zip file.

2.17.4 Response Messages

On success (status=200), the body of the response will contain the binary stream for the image.

On failure, the response provides the following information:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.18 The Get File Service

This service retrieves a file from the Docmosis file storage area associated with your account. The files that can be found in storage are those uploaded using the Put File Service. The List Files Service can list the files available for download.

> **The file storage feature of Docmosis is not available for all account types.**
>
> **We recommend AWS S3 storage using your own AWS account instead.**

2.18.1 Service URL

```
/getFile
```

2.18.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.18.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`fileName`	The name of the image file you require. This parameter can be specified multiple times to download multiple images (up to 100) in a zip file.

2.18.4 Response Messages

On success (status=200), the body of the response will contain the binary stream for the file.

On failure, the response provides the following information:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.19 The Put File Service

This service stores a file in the Docmosis file storage area associated with your account. This allows you to store files of any type in your account.

The file storage feature of Docmosis is not available for all account types.

We recommend AWS S3 storage using your own AWS account instead.

2.19.1 Service URL

```
/putFile
```

2.19.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.19.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`file`	The file stream to upload.
`fileName`	An optional overriding file name which may also include a path.
`contentType`	An optional setting for the content-type of this file. Docmosis will attempt to determine the content type if not specified.
`metaData`	An option string of information to store with this file that can be retrieved with the file later.

2.19.4 Response Messages

The body of the response contains a success indicator and error details if the request failed:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a

Field	Definition
	successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.20 The List Files Service

This service lists the files in the Docmosis file storage area associated with your account.

The file storage feature of Docmosis is not available for all account types.

We recommend AWS S3 storage using your own AWS account instead.

2.20.1 Service URL

```
/listFiles
```

2.20.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.20.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`folder`	An option starting folder (path). If not specified, all files in your storage will be listed. The returned names are always relative to the folder.
`includeSubFolders`	An optional specification as to whether you would like the list to include items within sub-folders. Defaults to false. "y", "yes" and "true" are all positive indicators.
`includeMetaData`	If "y", "yes" or "true" meta data for each file will be included in the results. Defaults to "false".

2.20.4 Response Messages

The response includes the normal success indicator and messages as well as a JSON object containing the list of files.

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.
`storedFileListStale`	"true" or "false". If Docmosis detects changes to the stored file list that are in progress (such as updates or deletions) this flag will be set to "true" to indicate the list is not necessarily up to date. This is only ever expected to be "true" for a short period after deletes or updates.
`storedFileList`	The list of images in JSON format having attributes for each file: `name` - the file name `lastModifiedMillisSinceEpoch` - last modified in milliseconds `lastModifiedISO8601` - last modified yyyy-MM-dd'T'HH:mm:ssZ `sizeBytes` - the size in bytes `metaData` - the metadata stored with the file (if requested)

2.21 The Delete Files Service

This service deletes files stored in the Docmosis file storage area associated with your account.

The file storage feature of Docmosis is not available for all account types.

We recommend AWS S3 storage using your own AWS account instead.

2.21.1 Service URL

```
/deleteFiles
```

2.21.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.21.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`path`	The name of the file or folder.
`includeSubFolders`	If "y", "yes" or "true" all files within the given path are deleted also. Default is false.

2.21.4 Response Messages

The service responds with a simple indication of success or failure using the standard structure:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.22 The Rename Files Service

This service allows files and folders in the Docmosis file storage area associated with your account to be renamed.

The file storage feature of Docmosis is not available for all account types.

We recommend AWS S3 storage using your own AWS account instead.

2.22.1 Service URL

```
/renameFiles
```

2.22.2 Content-Type

The content-type for the call may be "`application/x-www-form-urlencoded`" or "`multipart/form-data`".

2.22.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`fromPath`	The original name of the file or folder.
`toPath`	The new name for the file or folder.

2.22.4 Response Messages

The service responds with a simple indication of success or failure using the standard structure:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.23 The Convert Service

The convert service allows files to be converted between formats. The process is a simple conversion with no concept of templates and data. It applies to spreadsheet, presentation and drawing types of document.

2.23.1 Service URL

```
/convert
```

2.23.2 Content-Type

The content-type for the call is " `multipart/form-data`".

2.23.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`file`	The file to convert.
`outputName`	The name of the new file to create. The extension is used to determine the type of file to create. For example, “result.pdf” causes a PDF document to be created.

2.23.4 Response Messages

The service responds with a simple indication of success or failure using the standard structure:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

2.24 The Get Render Tags Service

The get render tags service allows statistics to be retrieved on renders that were tagged with user-defined phrases (“tags”). The statistics include page counts and document counts that have been collected against the tags, aggregated monthly. This may be useful for reporting the level of activity of a group of users, or a feature in your application.

2.24.1 Service URL

```
/getRenderTags
```

2.24.2 Content-Type

The content-type for the call may be "`multipart/form-data`" or "`application/x-www-form-urlencoded`".

2.24.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`Tags`	The tags to query. This can be a single tag or a list of tags separated by the ; (semicolon) character. This parameter is mandatory.
`year`	The year on which to report statistics. Defaults to the current year.
`month`	The month on which to report statistics (1=Jan). Defaults to the current month.
`nMonths`	The number of months on which to report statistics. Defaults to 1. If more than one month is being reported, the months prior to the specified year and month are included. In other words, this call always reports up to the specified month.
`padBlanks`	If true (or 'y'), zero values will be included where no data exists. This may make parsing the returned result easier since it will always contain values for the tags requested over the given time period by padding the data with zero-values as required. Defaults to false.

2.24.4 Response Messages

The service responds with a JSON structure as follows:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. In the case of an error this will be a long error message. It may be blank.

Field	Definition
renderTags	An array of objects as with values as follows: year – the year month – the month number tags – an array of objects as follows: name – the tag countPages – the number of pages rendered for this tag in the year and month countDocuments – the number of documents rendered against this tag in the year and month For each tag queried, there will be an entry for that tag in the year and month. There will also be a "combined" result showing the statistics for the combined set of tags queried. This combined tag will report the stats that match all of the tags specified in the call to the getRenderTags request (that is, it will return the stats only for renders that included all the tags).

For example, if a call is made with the following parameters:

```
tags:  abc;def

padBlanks: true

year: 2017

month: 9

nMonths: 2
```

Then the following result shows and example JSON response:

```
{
  "succeeded": true,
  "renderTags": [
    {
      "year": "2017",
      "month": "8",
      "tags": [
        {
          "name": "abc",
          "countPages": "0",
          "countDocuments": "0"
        },
        {
```

```
            "name": "def",
            "countPages": "0",
            "countDocuments": "0"
          },
          {
            "name": "abc;def",
            "countPages": "0",
            "countDocuments": "0"
          }
        ]
      },
      {
        "year": "2017",
        "month": "9",
        "tags": [
          {
            "name": "abc",
            "countPages": "6",
            "countDocuments": "4"
          },
          {
            "name": "def",
            "countPages": "4",
            "countDocuments": "2"
          },
          {
            "name": "abc;def",
            "countPages": "2",
            "countDocuments": "1"
          }
        ]
      }
    ]
  }
```


In the above output, we can see that we have two objects in the array of “renderTags”, one for month 8 (Aug) and one for month 9 (Sep). Each object in the array contains the stats for each requested tag (“abc” and “def”) and the combined tag (“abc;def”). There is no data for August but because padBlanks is true, the data is filled out with zeros.

In September, we can see that 6 pages were generated by renders with tag “abc”, 4 pages with tag “def” and 2 pages with both “abc” and “def” tags.

2.25 The Get Sample Data Service

The get sample data service allows sample data to be generated for a template based on the current structures in the template. The sample data can be created in JSON or XML format which can then be fed back to the render service to generate populated documents.

The service creates values like “value1”, “value2” for each field element.

If the template has an error in it, Docmosis will generate a blank data set.

2.25.1 Service URL

```
/getSampleData
```

2.25.2 Content-Type

The content-type for the is "`multipart/form-data`" or "`application/x-www-form-urlencoded`".

2.25.3 Request Parameters

Field	Definition
`accessKey`	Your access key.
`templateName`	The name of the template for which to create sample data.
`format`	If blank or “json”, JSON format data will be returned. Otherwise XML format data will be returned.

2.25.4 Response Messages

The service responds with a JSON structure as follows:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case of an error this will be a short error message. It may be blank in the case of a successful operation.
`longMsg`	A more descriptive message about the result. It may be blank, or, In the case of an error, a long error message.
`templateSampleData`	JSON or XML formatted sample data that can be used to

Field	Definition
	populate the template.

2.26 The Ping Service

The ping service provides a direct check that the Docmosis Cloud services are online and there is at least one Docmosis server listening. This is useful for diagnostics and for monitoring purposes. The response is empty, and the http response code “200” is the success indicator.

2.26.1 Service URL

```
/ping
```

2.26.2 Content-Type

The content-type is not specified, since the call takes no parameters.

2.26.3 Request Parameters

None.

2.26.4 Response Messages

There is no response body returned.

2.27 The Account Ready Service

The account ready service provides the ability to check whether an account is ready to service document requests. This includes checking whether the account is active and within quota limits. Note: production accounts may be over quota and still operational, in which case this method will indicate the account is ready, even when over quota.

2.27.1 Service URL

```
/accountReady
```

2.27.2 Content-Type

The content-type for the is "`multipart/form-data`" or "`application/x-www-form-urlencoded`".

2.27.3 Request Parameters

Field	Definition
`accessKey`	Your access key.

2.27.4 Response Codes

This service is intended for monitoring / confirmation of ready state. The return codes are:

200 – account is ready

400 – account is not ready

2.27.5 Response Messages

The service responds with a JSON structure as follows:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the result. In the case where the account is not ready, this message will indicate why.

2.28 The Account Summary Service

The account summary service returns details about an account including plan information, account status and quotas. Note: this service returns a 200 response even if the account is not ready which is different from the accountReady service.

2.28.1 Service URL

```
/accountSummary
```

2.28.2 Content-Type

The content-type for the is "`multipart/form-data`" or "`application/x-www-form-urlencoded`".

2.28.3 Request Parameters

Field	Definition
`accessKey`	Your access key.

2.28.4 Response Messages

The service responds with a JSON structure as follows:

Field	Definition
`succeeded`	"true" or "false".
`shortMsg`	A short message about the ready-status of the account
`accountSummary`	A JSON structure providing details of the account: accountSummary: ready – whether the account is ready to create documents accountDetails: isActivated is the account activated isDisabled has the account been disabled pageQuota: used – pages rendered this month quota – monthly page quota pctUsed – quota used as a percentage pctUsedStr – quota used as a string isHardLimited – whether the account is blocked because of over-quota plan: name – the name of the plan.

An example response is:

```
"accountSummary":{
  "ready":"true",
  "accountDetails":{
    "isActivated":"true",
```


```
    "isDisabled":"false"
  },
  "pageQuota":{
     "used":"467",
     "quota":"230",
     "pctUsed":"203",
     "pctUsedStr":"203%",
     "isHardLimited":"false"
  },
  "plan":{
    "name":"Free Trial"
  }
}
```